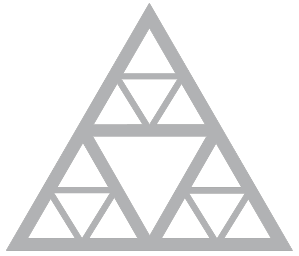
Introduction à la programmation en C++

Préliminaires

Nicolas Audebert

14 septembre 2018



École des Ponts
ParisTech

Plan de la séance

Avant toute chose

Supports de cours

- Site du cours : `http://imagine.enpc.fr/~monasse/Info/`
- Planches : `https://nicolas.audebert.at/teaching/`
- Le polycopié « La programmation pour les élèves-ingénieurs »

Organisation

- 12 séances
- Cours magistral de 8h30 à 9h30 puis TP de 9h45 à 11h15
- Travaux pratiques à rendre chaque semaine sur **Educnet**

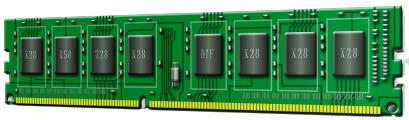
Plan de la séance

Composants de l'ordinateur

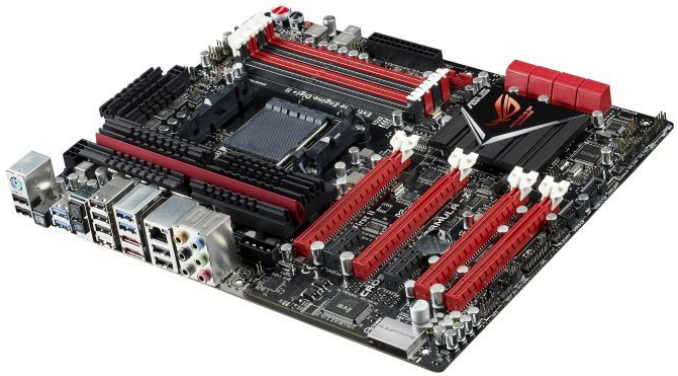
Stockage (disque dur, SSD…)

Processeur (CPU)

Périphériques



Mémoire vive (RAM)



Carte mère

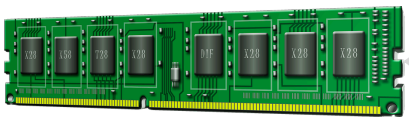
Carte graphique (GPU)

Composants de l'ordinateur

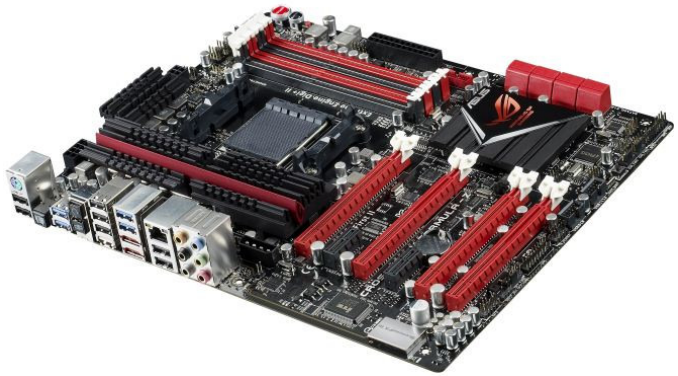
Stockage (disque dur, SSD…)

Processeur (CPU)

Périphériques



Mémoire vive (RAM)



Carte mère

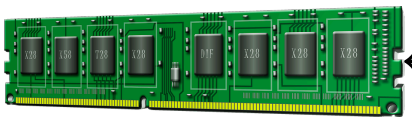
Carte graphique (GPU)

Composants de l'ordinateur

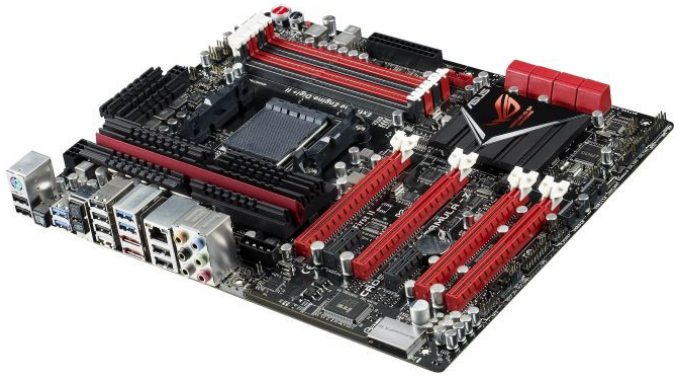
Stockage (disque dur, SSD…)

Processeur (CPU)

Périphériques



Mémoire vive (RAM)



Carte mère

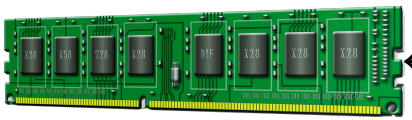
Carte graphique (GPU)

Composants de l'ordinateur

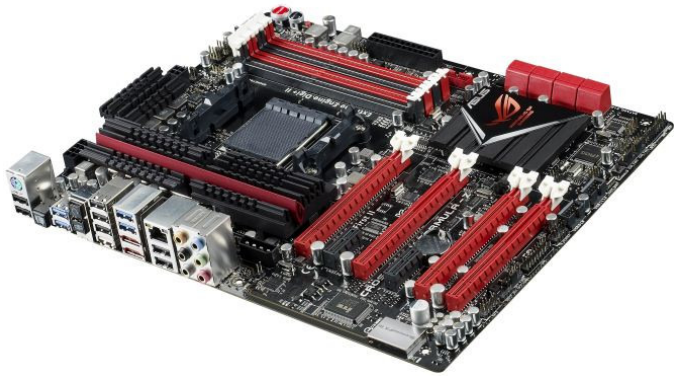
Stockage (disque dur, SSD…)

Processeur (CPU)

Périphériques



Mémoire vive (RAM)



Carte mère

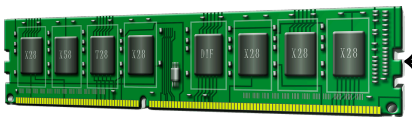
Carte graphique (GPU)

Composants de l'ordinateur

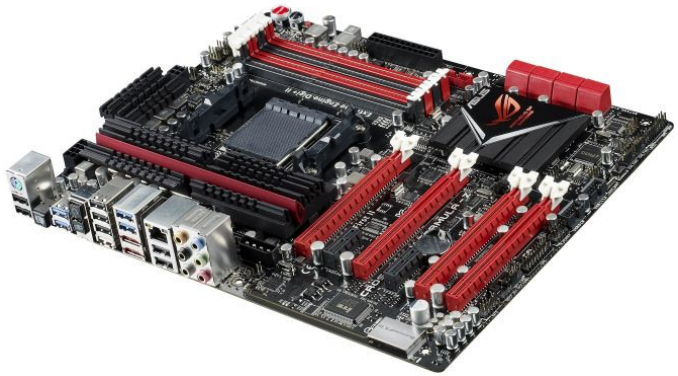
Stockage (disque dur, SSD…)

Processeur (CPU)

Périphériques



Mémoire vive (RAM)



Carte mère

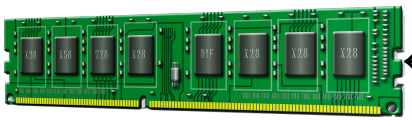
Carte graphique (GPU)

Composants de l'ordinateur

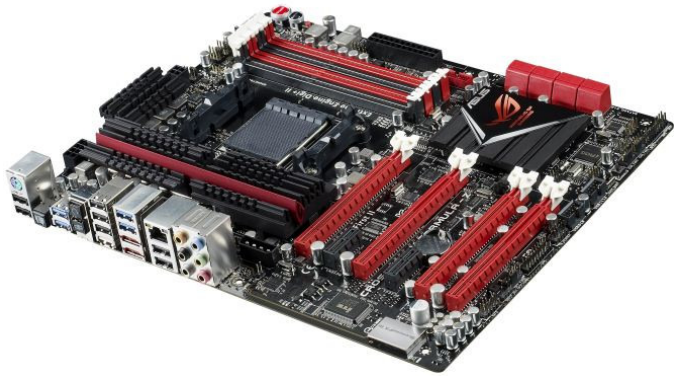
Stockage (disque dur, SSD…)

Processeur (CPU)

Périphériques



Mémoire vive (RAM)



Carte mère

Carte graphique (GPU)

Système d'exploitation

OS (operating system)

Le système d'exploitation orchestre les ressources de l'ordinateur. Il ordonne les tâches du processeur en fonction de leur priorité, gère la mémoire vive et le stockage et dirige la communication avec les périphériques.

→ **Exemples :** Windows, Linux, MacOS, Android, BSD…

Qu'est-ce qu'un langage de programmation ?

Un ensemble de symboles (syntaxe) et de règles d'écriture (grammaire) permettant de créer un programme.

- Langage machine (**assembleur**)
- Langage procédural (FORTRAN, **C**...)
- Langage objet (**C++**, Python, Java...)
- Langage fonctionnel (Lisp, **OCaml**...)
- Langage exotique (Ook, **LOLCODE**...)

```
section .data
     helloMsg: db 'Hello world!',10
     helloSize: equ $-helloMsg
section .text
     global _start
_start:
     mov eax,4
     mov ebx,1
     mov ecx, helloMsg
     mov edx, helloSize
     int 80h
     mov eax,1
     mov ebx,0
     int 80h
```

Langages de programmation

Qu'est-ce qu'un langage de programmation ?

Un ensemble de symboles (syntaxe) et de règles d'écriture (grammaire) permettant de créer un programme.

- Langage machine (**assembleur**)
- Langage procédural (FORTRAN, **C**...)
- Langage objet (**C++**, Python, Java...)
- Langage fonctionnel (Lisp, **OCaml**...)
- Langage exotique (Ook, **LOLCODE**...)

```c
#include<stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

Langages de programmation

Qu'est-ce qu'un langage de programmation ?

Un ensemble de symboles (syntaxe) et de règles d'écriture (grammaire) permettant de créer un programme.

- Langage machine (**assembleur**)
- Langage procédural (FORTRAN, **C**...)
- Langage objet (**C++**, Python, Java...)
- Langage fonctionnel (Lisp, **OCaml**...)
- Langage exotique (Ook, **LOLCODE**...)

```cpp
#include <iostream>
using namespace std;

int main(){
  cout << "Hello world!" << endl;
  return 0;
}
```

Langages de programmation

Qu'est-ce qu'un langage de programmation ?

Un ensemble de symboles (syntaxe) et de règles d'écriture (grammaire) permettant de créer un programme.

- Langage machine (**assembleur**)
- Langage procédural (FORTRAN, **C**...)
- Langage objet (**C++**, Python, Java...)
- Langage fonctionnel (Lisp, **OCaml**...)
- Langage exotique (Ook, LOLCODE...)

```
print_endline "Hello world!"
```

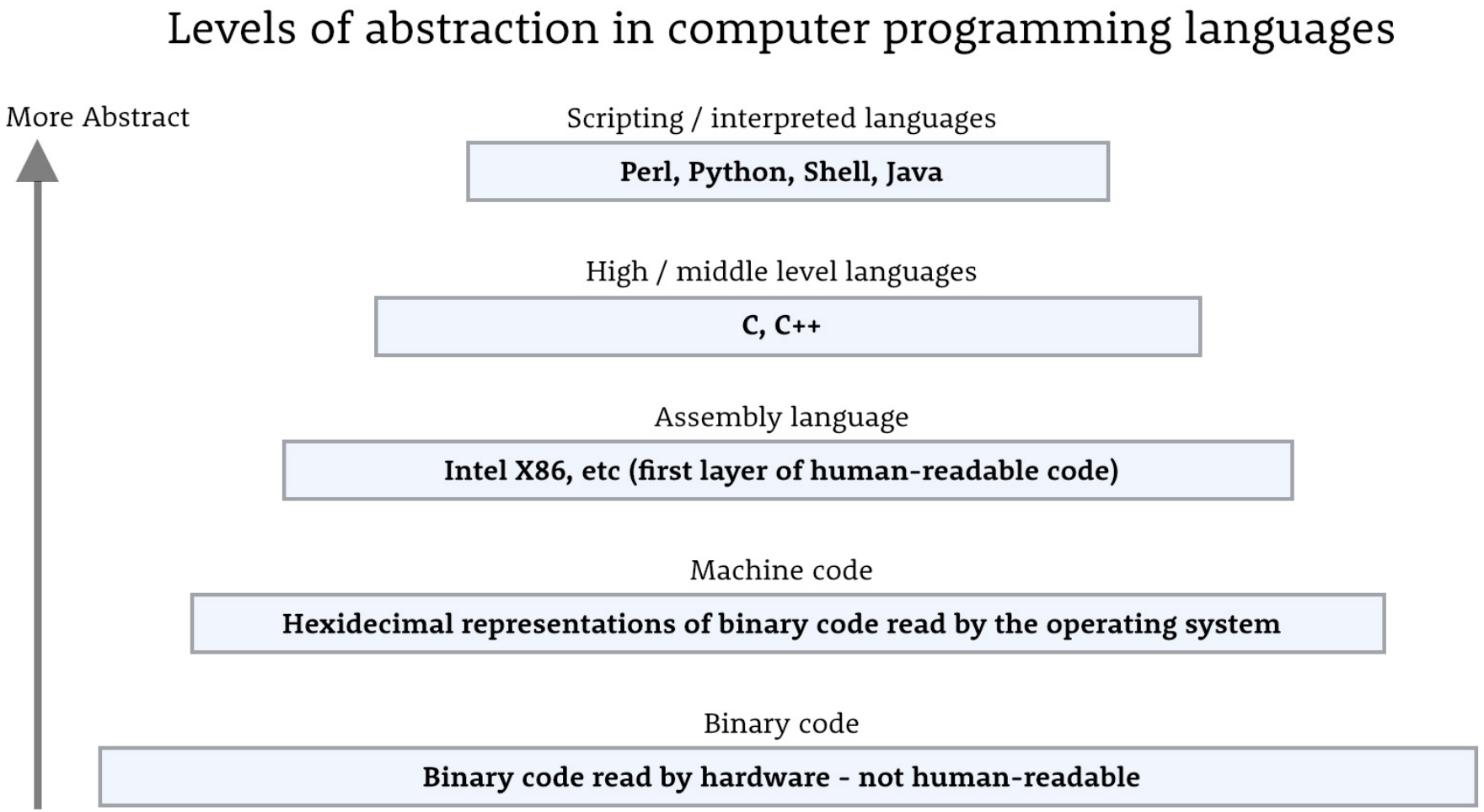
Qu'est-ce qu'un langage de programmation ?

Un ensemble de symboles (syntaxe) et de règles d'écriture (grammaire) permettant de créer un programme.

- Langage machine (**assembleur**)
- Langage procédural (FORTRAN, **C**...)
- Langage objet (**C++**, Python, Java...)
- Langage fonctionnel (Lisp, **OCaml**...)
- Langage exotique (Ook, **LOLCODE**...)

```
HAI
CAN HAS STDIO?
BTW affiche le message
VISIBLE "Hello world!"
KTHXBYE
```

Haut niveau ou bas niveau



Crédits image : YellowPencil

Compilé ou interprété ?

Langages compilés

- Le **compilateur** traduit à l'avance le code en instructions machines.
- Il transforme le code en un fichier exécutable (par exemple, `.exe` sous Windows).
- Exemples : C, C++, OCaml...

Langages interprétés

- L'**interpréteur** traduit le code à la volée.
- Le code n'a pas besoin d'être transformé, mais l'interpréteur est indispensable.
- Exemples : Javascript, Python, Ruby, PHP...

Pourquoi programmer en C++

- Un langage **complexe** : savoir programmer en C++ c'est savoir programmer dans tous les langages de la même famille (Java, C#, Go, Rust…).
- Un langage **complet**, qui permet de travailler à haut niveau d'abstraction ou à bas niveau, proche de l'architecture de la machine.
- Un langage **multi-paradigmes**, qui autorise la programmation objet, générique et impérative.
- Un des langages **les plus utilisés** (4e langage du Tiobe Index avec 7,4%, 4e langage le plus actif sur Github).
- Un langage industriel utilisé pour les microcontrôleurs, la simulation, les jeux vidéo, les bases de données…

Pourquoi programmer en C++

- Un langage **complexe** : savoir programmer en C++ c'est savoir programmer dans tous les langages de la même famille (Java, C#, Go, Rust...).
- Un langage **complet**, qui permet de travailler à haut niveau d'abstraction ou à bas niveau, proche de l'architecture de la machine.
- Un langage **multi-paradigmes**, qui autorise la programmation objet, générique et impérative.
- Un des langages **les plus utilisés** (4e langage du Tiobe Index avec 7,4%, 4e langage le plus actif sur Github).
- Un langage industriel utilisé pour les microcontrôleurs, la simulation, les jeux vidéo, les bases de données...

Pourquoi programmer en C++

- Un langage **complexe** : savoir programmer en C++ c'est savoir programmer dans tous les langages de la même famille (Java, C#, Go, Rust...).
- Un langage **complet**, qui permet de travailler à haut niveau d'abstraction ou à bas niveau, proche de l'architecture de la machine.
- Un langage **multi-paradigmes**, qui autorise la programmation objet, générique et impérative.
- Un des langages **les plus utilisés** (4e langage du Tiobe Index avec 7,4%, 4e langage le plus actif sur Github).
- Un langage industriel utilisé pour les microcontrôleurs, la simulation, les jeux vidéo, les bases de données...

Pourquoi programmer en C++

- Un langage **complexe** : savoir programmer en C++ c'est savoir programmer dans tous les langages de la même famille (Java, C#, Go, Rust...).
- Un langage **complet**, qui permet de travailler à haut niveau d'abstraction ou à bas niveau, proche de l'architecture de la machine.
- Un langage **multi-paradigmes**, qui autorise la programmation objet, générique et impérative.
- Un des langages **les plus utilisés** (4e langage du Tiobe Index avec 7,4%, 4e langage le plus actif sur Github).
- Un langage industriel utilisé pour les microcontrôleurs, la simulation, les jeux vidéo, les bases de données...

Pourquoi programmer en C++

- Un langage **complexe** : savoir programmer en C++ c'est savoir programmer dans tous les langages de la même famille (Java, C#, Go, Rust...).
- Un langage **complet**, qui permet de travailler à haut niveau d'abstraction ou à bas niveau, proche de l'architecture de la machine.
- Un langage **multi-paradigmes**, qui autorise la programmation objet, générique et impérative.
- Un des langages **les plus utilisés** (4e langage du Tiobe Index avec 7,4%, 4e langage le plus actif sur Github).
- Un langage industriel utilisé pour les microcontrôleurs, la simulation, les jeux vidéo, les bases de données...

Différences entre Python et C++

```python
# Python
a = 0.
b = 5
for i in range(10):
    a = b * i + 0.1
    print(a)
print((a + b) / 2)
```

```cpp
// C++
float a = 0
int b = 5
for(int i = 0; i < 10; i++){
    a = b * i + 0.1;
    cout << a;
}
cout << (a + b) / 2;
```

- Le Python est un langage interpreté, le C++ est un langage compilé
- Le C++ sera en général plus rapide que le Python
- La délimitation des blocs se fait par des accolades {}
- Chaque instruction se termine par un ; ← indispensable
- Les variables vivent dans le bloc où elles ont été créées
- Le type des variables est annoncé à leur déclaration

Différences entre Python et C++

```python
# Python
a = 0.
b = 5
for i in range(10):
    a = b * i + 0.1
    print(a)
print((a + b) / 2)
```

```cpp
// C++
float a = 0
int b = 5
for(int i = 0; i < 10; i++){
    a = b * i + 0.1;
    cout << a;
}
cout << (a + b) / 2;
```

- Le Python est un langage interpreté, le C++ est un langage compilé
- Le C++ sera en général plus rapide que le Python
- La délimitation des blocs se fait par des accolades {}
- Chaque instruction se termine par un ; ← indispensable
- Les variables vivent dans le bloc où elles ont été créées
- Le type des variables est annoncé à leur déclaration

Différences entre Python et C++

```python
# Python
a = 0.
b = 5
for i in range(10):
    a = b * i + 0.1
    print(a)
print((a + b) / 2)
```

```cpp
// C++
float a = 0
int b = 5
for(int i = 0; i < 10; i++){
    a = b * i + 0.1;
    cout << a;
}
cout << (a + b) / 2;
```

- Le Python est un langage interpreté, le C++ est un langage compilé
- Le C++ sera en général plus rapide que le Python
- La délimitation des blocs se fait par des accolades {}
- Chaque instruction se termine par un ; ← indispensable
- Les variables vivent dans le bloc où elles ont été créées
- Le type des variables est annoncé à leur déclaration

Différences entre Python et C++

```python
# Python
a = 0.
b = 5
for i in range(10):
    a = b * i + 0.1
    print(a)
print((a + b) / 2)
```

```cpp
// C++
float a = 0
int b = 5
for(int i = 0; i < 10; i++){
    a = b * i + 0.1;
    cout << a;
}
cout << (a + b) / 2;
```

- Le Python est un langage interpreté, le C++ est un langage compilé
- Le C++ sera en général plus rapide que le Python
- La délimitation des blocs se fait par des accolades {}
- Chaque instruction se termine par un ; ← indispensable
- Les variables vivent dans le bloc où elles ont été créées
- Le type des variables est annoncé à leur déclaration

Différences entre Python et C++

```python
# Python
a = 0.
b = 5
for i in range(10):
    a = b * i + 0.1
    print(a)
print((a + b) / 2)
```

```cpp
// C++
float a = 0
int b = 5
for(int i = 0; i < 10; i++){
    a = b * i + 0.1;
    cout << a;
}
cout << (a + b) / 2;
```

- Le Python est un langage interpreté, le C++ est un langage compilé
- Le C++ sera en général plus rapide que le Python
- La délimitation des blocs se fait par des accolades {}
- Chaque instruction se termine par un ; ← indispensable
- Les variables vivent dans le bloc où elles ont été créées
- Le type des variables est annoncé à leur déclaration

Différences entre Python et C++

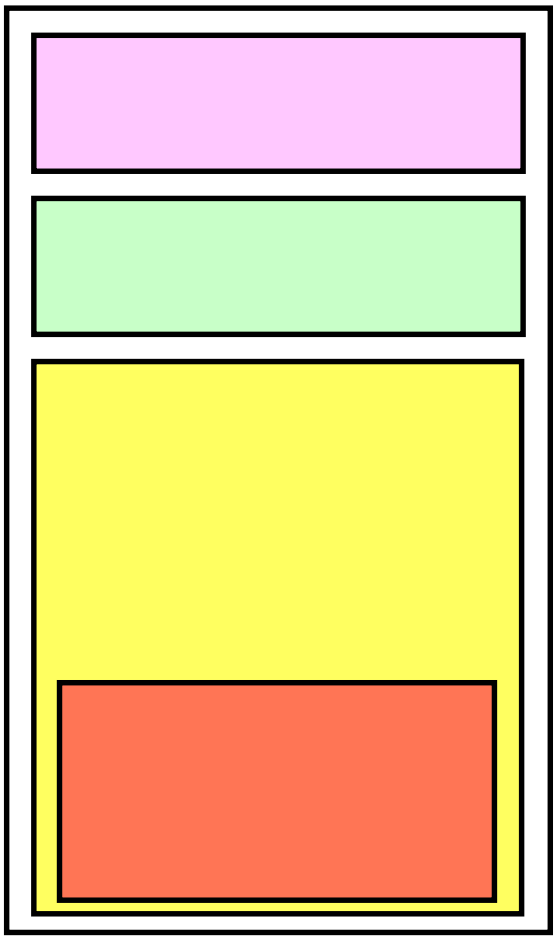
```python
# Python
a = 0.
b = 5
for i in range(10):
    a = b * i + 0.1
    print(a)
print((a + b) / 2)
```

```cpp
// C++
float a = 0
int b = 5
for(int i = 0; i < 10; i++){
    a = b * i + 0.1;
    cout << a;
}
cout << (a + b) / 2;
```

- Le Python est un langage interpreté, le C++ est un langage compilé
- Le C++ sera en général plus rapide que le Python
- La délimitation des blocs se fait par des accolades {}
- Chaque instruction se termine par un ; ← indispensable
- Les variables vivent dans le bloc où elles ont été créées
- Le type des variables est annoncé à leur déclaration

Plan de la séance

Structure des programmes



Inclusion des librairies

Définition des noms de domaine

Les fonctions (plus tard)

Le main (fonction particulière)

Hello world

```cpp
// commentaire sur une ligne
#include <iostream>
/* commentaires par
 * bloc */

// utilisation de l'espace de nom de la librairie standard
using namespace std;

int main()
{
    cout << "Hello world!"; // écriture de "Hello world!"
    cout << endl; // passage à la ligne

    cout << "HelloWorld" << endl; // les deux en même temps

    return 0;
}
```

Détails du programme : librairies

```
#include <iostream>
```

Appeler des libraires donne accès à des fonctions pré-existantes.
C'est l'équivalent du `import ...` en Python.

`<iostream>` fait partie de la Standard Template Library (STL).
`<iostream>` permet de gérer les affichages à l'écran et de récupérer des entrées clavier.

Détails du programme : nom de domaine

```
using namespace std;
```

En C++, les librairies peuvent avoir un nom de domaine (équivalent du nom de module en Python).

Définir le nom de domaine permet de ne pas le répéter avec chaque fonction appelée, de façon similaire à `from ... import *` en Python.

Exemple : `std::cout << std::endl;` se raccourcit en `cout << endl;` .

Détails du programme : la fonction main

```
int main(){
    cout << "HelloWorld";
    cout << endl;
    cout << "HelloWorld" << endl;
    return 0;
}
```

La fonction `main()` est le point d'entrée du programme. Lorsque l'exécutable est lancé, c'est la fonction `main()` qui est jouée. Cette fonction est **obligatoire**.

Attention

Contrairement à Python, il n'est pas possible d'écrire des instructions en dehors de la fonction `main` , celles-ci ne seront jamais exécutées.

Détails du programme : la fonction main

```
int main(){ // type : int, nom : main, arguments : rien
```

`int` est le type attendu du résultat de la fonction (ici un entier).

`main` est le nom de la fonction.

Les instructions formant le corps de la fonction sont comprises entre des accolades.

```
    ...
    return 0; // valeur de retour lorsque tout s'est bien passé
}
```

`main()` renvoie un entier, ici 0.

Détails du programme : le code

```
cout << "Hello world!";
cout << endl;
cout << "Hello world!" << endl;
```

`cout` : ouvre une communication permettant d'écrire à l'écran.

`<<` : transfère l'élément de droite vers l'élément de gauche.

`"Hello world!"` est une chaîne de caractères.

`endl` : caractère spécial de passage à la ligne.

```
using namespace std;
cout << "Hello world!";
cout << endl;
cout << "Hello world!" << endl;
```

Quelques règles pour bien démarrer

- Le code s'écrit **toujours** dans une fonction (`main` pour l'instant).
- Mettre en forme le code en l'indentant est **obligatoire**, mais pas nécessaire.
- Il y a **une et une seule** fonction `main()` par programme.
- Commenter son programme permet de le relire plus facilement.

Plan de la séance

Environnements de développement

Integrated Development Environment

L'Environnement de Développement Intégré est un logiciel à destination des développeurs et développeuses dont les fonctionalités aident à programmer.

Il existe de nombreux IDE, spécialisés pour un ou plusieurs langages :

- **C++** : QtCreator, Eclipse, Visual Studio, KDevelop, XCode, CodeBlocks...
- Python : Spyder, WingIDE, PyCharm...
- ...

QtCreator

QtCreator est l'IDE que nous utiliserons dans ce cours.

- Multiplateforme (Windows, Linux, OS X)
- Relativement simple à utiliser
- Débogueur intégré
- Autocomplétion
- Gestion de Cmake

Bibliothèque Imagine++

La bibliothèque Imagine++ est une bibliothèque de fonctions graphiques développée à l'ENPC.
Elle contient notamment :

- des fonctions pour l'affichage graphique (images, formes géométriques...),
- la gestion du clavier et de la souris,
- des outils de gestion des fenêtres (création, destruction, affichage temps réel...),
- le nécessaire pour l'agèbre linéaire de base (matrices, vecteurs...).

IDE et projets

Chaque IDE stocke les informations relative à un projet dans un format spécifique :

- L'emplacement des fichiers de code
- Les emplacements des librairies

Il est parfois fastidieux de construire un projet.

Pour y remédier on utilise un **moteur de production**.

CMake

CMake est **moteur de production**, un logiciel multiplateforme permettant de générer des projets indifféremment de l'IDE utilisé.

Utilisation

- Interface utilisateur : Makefiles (Unix), Visual Studio (Windows), Xcode, Eclipse...
- Directement dans l'IDE : QTCreator, KDevelop...

CMakeLists.txt

Exemple de fichier **`CMakeLists.txt`** décrivant un projet :

```
# CMakeLists.txt
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)

#Inclusion des modules (ici Imagine++)
FILE(TO_CMAKE_PATH "$ENV{IMAGINEPP_ROOT}" d)
IF(NOT EXISTS "${d}")
  MESSAGE(FATAL_ERROR "Error: IMAGINEPP_ROOT=" "${d}")
ENDIF(NOT EXISTS "${d}")
SET(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "${d}/CMake")
FIND_PACKAGE(Imagine)

# Création d'un projet "monTP"
PROJECT(monTP)
# Ajout d'un exécutable "monTP"
add_executable(monTP monfichier.cpp)
# Utilisation de Imagine++ (partie Graphics) pour l'exécutable "monTP"
ImagineUseModules(monTP Graphics)
```

Déboguer avec l'IDE

Le débogueur est outil important de l'IDE par rapport au bloc-notes + ligne de commande.

Il permet d'exécuter le programme pas à pas, de l'arrêter à n'importe quel endroit et d'inspecter le contenu des variables. Cela permet de détecter plus simplement où se trouvent les erreurs dans le code.

Travaux pratiques

Aujourd'hui :

- Prise en main d'un IDE,
- Compilation, exécution, débogage,
- Ce TP n'est pas à rendre.

Installation de QtCreator et Imagine++ sur vos machines :

- À faire chez vous...
- ou aux séances d'aide : cet après-midi 14/09 de 13h30 à 18h en F206/F207 et lundi 17/09 de 16h45 à 18h45 en F107.